Machine Learning

Exam 13/01/2022

Student: Matteo Donati
Registration no.: 0001032227
E-mail: matteo.donati10@studio.unibo.it (mailto:matteo.donati10@studio.unibo.it)

Libraries

The main libraries are imported.

In [205]:

```
# Importing pandas.
import pandas as pd

# Importing seaborn.
import seaborn as sns

# Importing numpy.
import numpy as np

# Importing OrdinalEncoder.
from sklearn.preprocessing import OrdinalEncoder

# Importing ColumnTransformer.
from sklearn.compose import ColumnTransformer

# Importing train_test_split.
from sklearn.model_selection import train_test_split

# Importing DecisionTreeClassifier.
from sklearn.tree import DecisionTreeClassifier

# Importing KNeighborsClassifier.
from sklearn.neighbors import KNeighborsClassifier

# Importing GridSearchCV.
from sklearn.model_selection import GridSearchCV

# Importing confusion_matrix.
from sklearn.metrics import confusion_matrix

# Importing classification_report.
from sklearn.metrics import classification_report
```

1. Data inspection

The dataset is loaded and inspected.

In [206]:

```
# Filename.
filename = "exam2022_01_13.csv"

# Loading the data.
df = pd.read_csv(filename, sep = ",")
```

In [207]:

```
# Printing the head of df.
df.head()
```

Out[207]:

	language	X1	X2	X3	X4	X5	X6	X7	
0	ES	7.071476	-6.512900	7.650800	11.150783	-7.657312	12.484021	-11.709772	3.426
1	ES	10.982967	-5.157445	3.952060	11.529381	-7.638047	12.136098	-12.036247	3.491
2	ES	7.827108	-5.477472	7.816257	9.187592	-7.172511	11.715299	-13.847214	4.574
3	ES	6.744083	-5.688920	6.546789	9.000183	-6.924963	11.710766	-12.374388	6.169
4	ES	5.836843	-5.326557	7.472265	8.847440	-6.773244	12.677218	-12.315061	4.416

In [208]:

```
# The shape of df is printed.
print("Number of rows: {}, number of columns: {}".format(df.shape[0], df.shape[1]))
```

```
Number of rows: 329, number of columns: 13
```

In [209]:

```
# Checking which columns contain numerical data and which contain categorical data.
df.dtypes
```

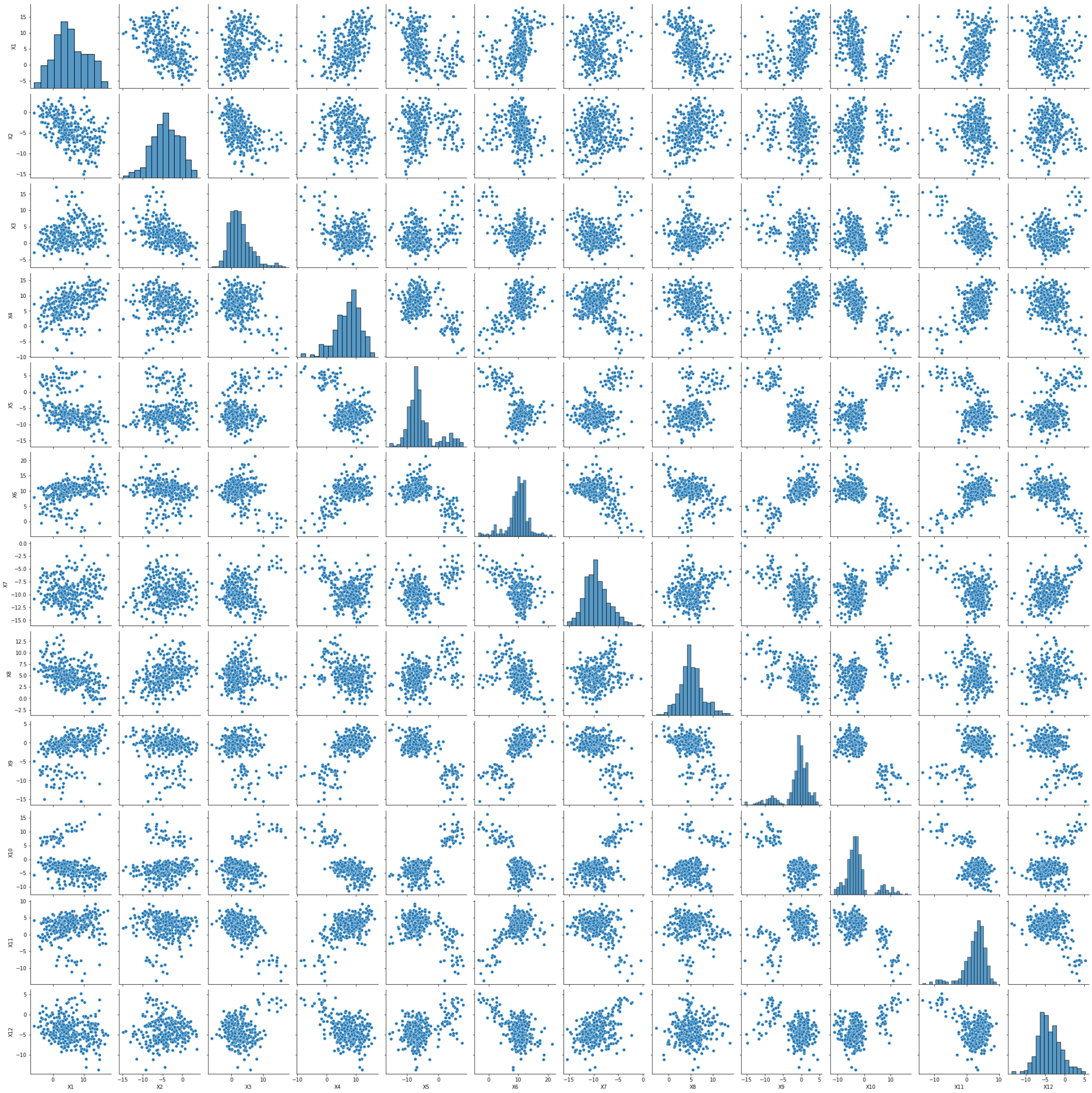
Out[209]:

```
language     object
X1          float64
X2          float64
X3          float64
X4          float64
X5          float64
X6          float64
X7          float64
X8          float64
X9          float64
X10         float64
X11         float64
X12         float64
dtype: object
```

Every column contains numerical data, except for the labels column. The pairplot of each numerical column is plotted. In particular, the main diagonal of such pairplot contains the histogram of each numerical attribute.

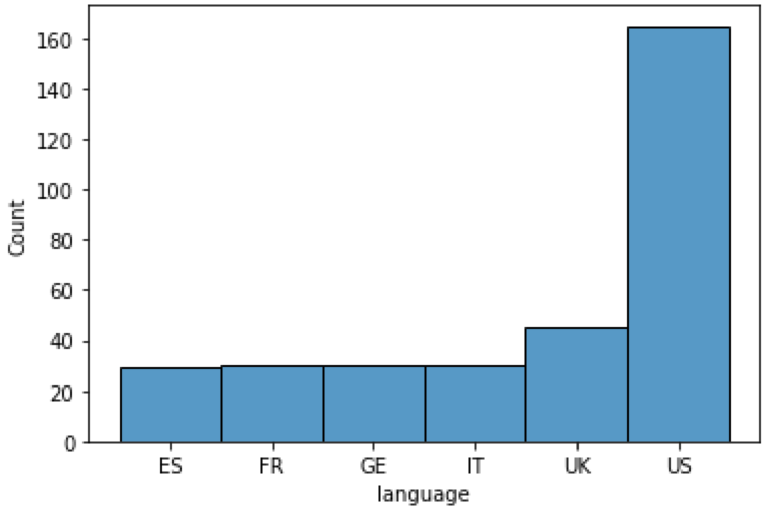
In [210]:

```
# Plotting the pairplot of numerical data.
sns.pairplot(df.drop(["language"], axis = 1));
```



In [211]:

```
# Plotting the histogram of the "language" column.
sns.histplot(df["language"]);
```



2. Removal of NaNs

The rows which contain NaN values are dropped.

In [212]:

```
# Checking if there are NaNs.
if df.isnull().values.any():

    # Removing rows with NaNs.
    df = df.dropna(axis = 0)

    # The shape of df is printed.
    print("Number of rows: {}, number of columns: {}".format(df.shape[0], df.shape[1]))
```

```
Number of rows: 321, number of columns: 13
```

Data preparation

In this part, the categorical attribute `language` is converted into a numerical attribute using an ordinal encoder.

In [213]:

```
# Defining the list of categorical attributes.
categorical_attributes = df.dtypes.loc[df.dtypes == "object"].index.values

# Printing the list of ordinal attributes.
print("Categorical attributes' columns: {}.".format(categorical_attributes))
```

```
Categorical attributes' columns: ['language'].
```

In [214]:

```
# Defining the categorical attributes transformer.
categorical_transformer = OrdinalEncoder(dtype = np.int32)

# Defining the list containing the ("label", "transformer", "list of columns") tuples.
transformer = [("ordinal", categorical_transformer, categorical_attributes)]

# Defining the preprocessor by passing as input the defined transformer.
preprocessor = ColumnTransformer(transformer, remainder = "passthrough")

# Computing the processed dataframe.
df_p = preprocessor.fit_transform(df)

# Converting df_p into a DataFrame object.
df_p = pd.DataFrame(df_p)

# Printing the first lines of the new dataframe.
df_p.head()
```

Out[214]:

	0	1	2	3	4	5	6	7	8	
0	0.0	7.071476	-6.512900	7.650800	11.150783	-7.657312	12.484021	-11.709772	3.426596	
1	0.0	10.982967	-5.157445	3.952060	11.529381	-7.638047	12.136098	-12.036247	3.491943	
2	0.0	7.827108	-5.477472	7.816257	9.187592	-7.172511	11.715299	-13.847214	4.574075	-
3	0.0	6.744083	-5.688920	6.546789	9.000183	-6.924963	11.710766	-12.374388	6.169879	-
4	0.0	5.836843	-5.326557	7.472265	8.847440	-6.773244	12.677218	-12.315061	4.416344	

The `language` column has been transformed into the `0` column of the dataframe.

In [215]:

```
# Defining the X matrix.
X = df_p.drop([0], axis = 1)

# Defining the y vector.
y = df_p[0]

# Printing the X shape.
print("X's shape: {}.".format(X.shape))

# Printing the y shape.
print("y's shape: {}.".format(y.shape))
```

```
X's shape: (321, 12).
y's shape: (321,).
```

In [216]:

```
# Setting the random state.
random_state = 42

# Splitting into training set and test set.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = random_state)
```

Models preparation

The two models, and their respective parameters, are defined.

In [217]:

```
# Setting the parameters to be explored by GridSearchCV for the decision tree model.
tuned_param_dt = [{"max_depth": list(range(1, 16))}]

# Setting the parameters to be explored by GridSearchCV for the k nearest neighbor model.
tuned_param_knn =[{"n_neighbors": list(range(1, 16))}]

# Defining the models to be fitted.
models = {
    "Model1": {"name": "Decision Tree",       "estimator": DecisionTreeClassifier(), "param"
    "Model2": {"name": "K Nearest Neighbor", "estimator": KNeighborsClassifier(),   "param"
}

# Defining the list of scores to be explored.
score = "recall_macro"
```

Auxiliary function for the computation of the classification report

The `print_result` function is defined.

In [218]:

```
# Defining the function which takes as input the fitted model and returns the classificatio
def print_results(name, model, y_test, y_pred):

    # Printing the type of model.
    print("Model name: {}.".format(name))

    # Printing the grid scores obtained during training.
    print("\nGrid scores:")

    # Iterating over the grid rows.
    for mean, std, params in zip(model.cv_results_["mean_test_score"], model.cv_results_["s

        # Printing one row of the grid.
        print("{:.3f} (+/- {:.3f}) for {}".format(mean, std * 2, params))

    # Printing the best parameters set.
    print("\nBest parameters set: {}.".format(model.best_params_))

    # Printing the detailed classification report.
    print("\nDetailed classification report:\n{}".format(classification_report(y_test, y_pr
```

3. Tuning of the hyper-parameters for the first model

The hyper-parameters of the first model are tuned.

In [219]:

```
# Activating the grid search.
clf = GridSearchCV(models["Model1"]["estimator"], models["Model1"]["param"], scoring = scor

# Fitting the model.
clf.fit(X_train, y_train);
```

4. Classification report for the first model

The function `print_results` is used to print the final report for the first model.

In [220]:

```
# Computing predictions with the test set.
y_pred = clf.predict(X_test)

# Printing the results.
print_results(models["Model1"]["name"], clf, y_test, y_pred)
```

```
Model name: Decision Tree.

Grid scores:
0.285 (+/- 0.080) for {'max_depth': 1}
0.324 (+/- 0.098) for {'max_depth': 2}
0.374 (+/- 0.198) for {'max_depth': 3}
0.460 (+/- 0.146) for {'max_depth': 4}
0.509 (+/- 0.169) for {'max_depth': 5}
0.573 (+/- 0.116) for {'max_depth': 6}
0.576 (+/- 0.078) for {'max_depth': 7}
0.595 (+/- 0.108) for {'max_depth': 8}
0.568 (+/- 0.106) for {'max_depth': 9}
0.573 (+/- 0.138) for {'max_depth': 10}
0.574 (+/- 0.106) for {'max_depth': 11}
0.578 (+/- 0.118) for {'max_depth': 12}
0.556 (+/- 0.124) for {'max_depth': 13}
0.570 (+/- 0.124) for {'max_depth': 14}
0.567 (+/- 0.118) for {'max_depth': 15}

Best parameters set: {'max_depth': 8}.

Detailed classification report:
              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88         8
         1.0       0.45      0.71      0.56         7
         2.0       0.38      0.33      0.35         9
         3.0       0.50      0.55      0.52        11
         4.0       0.17      0.17      0.17         6
         5.0       0.83      0.75      0.79        40

    accuracy                           0.64        81
   macro avg       0.53      0.56      0.54        81
weighted avg       0.66      0.64      0.65        81
```

5. Confusion matrix for the first model

The confusion matrix for the first model is computed and printed.

In [221]:

```
# Printing the confusion matrix.
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, y_pred)))
```

```
Confusion matrix:
[[ 7  1  0  0  0  0]
 [ 0  5  0  1  1  0]
 [ 0  1  3  2  1  2]
 [ 0  0  2  6  1  2]
 [ 0  1  0  2  1  2]
 [ 1  3  3  1  2 30]]
```

6. Tuning of the hyper-parameters for the second model

The hyper-parameters of the second model are tuned.

In [222]:

```
# Activating the grid search.
clf = GridSearchCV(models["Model2"]["estimator"], models["Model2"]["param"], scoring = scor

# Fitting the model.
clf.fit(X_train, y_train);
```

7. Classification report for the second model

The function `print_results` is used to print the final report for the second model.

In [223]:

```
# Computing predictions with the test set.
y_pred = clf.predict(X_test)

# Printing the results.
print_results(models["Model2"]["name"], clf, y_test, y_pred)
```

```
Model name: K Nearest Neighbor.

Grid scores:
0.790 (+/- 0.086) for {'n_neighbors': 1}
0.753 (+/- 0.102) for {'n_neighbors': 2}
0.762 (+/- 0.077) for {'n_neighbors': 3}
0.751 (+/- 0.181) for {'n_neighbors': 4}
0.724 (+/- 0.193) for {'n_neighbors': 5}
0.734 (+/- 0.184) for {'n_neighbors': 6}
0.674 (+/- 0.156) for {'n_neighbors': 7}
0.643 (+/- 0.166) for {'n_neighbors': 8}
0.630 (+/- 0.154) for {'n_neighbors': 9}
0.621 (+/- 0.136) for {'n_neighbors': 10}
0.644 (+/- 0.157) for {'n_neighbors': 11}
0.639 (+/- 0.143) for {'n_neighbors': 12}
0.616 (+/- 0.155) for {'n_neighbors': 13}
0.589 (+/- 0.195) for {'n_neighbors': 14}
0.546 (+/- 0.199) for {'n_neighbors': 15}

Best parameters set: {'n_neighbors': 1}.

Detailed classification report:
              precision    recall  f1-score   support

         0.0       0.86      0.75      0.80         8
         1.0       0.88      1.00      0.93         7
         2.0       0.64      0.78      0.70         9
         3.0       0.89      0.73      0.80        11
         4.0       0.50      0.50      0.50         6
         5.0       0.90      0.90      0.90        40

    accuracy                           0.83        81
   macro avg       0.78      0.78      0.77        81
weighted avg       0.83      0.83      0.83        81
```

8. Confusion matrix for the second model

The confusion matrix for the second model is computed and printed.

In [224]:

```
# Printing the confusion matrix.
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, y_pred)))
```

```
Confusion matrix:
[[ 6  0  0  0  0  2]
 [ 0  7  0  0  0  0]
 [ 0  0  7  0  2  0]
 [ 0  0  1  8  1  1]
 [ 0  0  1  1  3  1]
 [ 1  1  2  0  0 36]]
```